Package 'logging'

July 2, 2014

Version 0.7-103

Date 2013-04-08

Title R logging package

Author Mario Frasca <mariotomo@gmail.com>

Maintainer Mario Frasca <mariotomo@gmail.com>

Description logging is a pure R package that implements the ubiquitous log4j package.

License GPL (>= 2)

Depends R (>= 2.12.0), methods

Suggests svUnit, XML

Repository CRAN

Repository/R-Forge/Project logging

Repository/R-Forge/Revision 103

Repository/R-Forge/DateTimeStamp 2013-04-08 09:57:38

Date/Publication 2013-04-12 08:30:00

NeedsCompilation no

R topics documented:

`logging-package` *a tentative logging package*

Description

a logging package emulating the python logging package.

What you find here behaves similarly to what you also find in Python. This includes hierarchic loggers, multiple handlers at each logger, the possibility to specify a formatter for each handler (one default formatter is given), same levels (names and numeric values) as Python's logging package, a simple logging.BasicConfig function to quickly put yourself in a usable situation...

This package owes a lot to my employer, r-forge, the stackoverflow community, Brian Lee Yung Rowe's futile package (v1.1) and the documentation of the Python logging package.

Details

Package:	logging
Version:	0.4-35
Date:	2010-01-14
License:	GPL (>=2)
Depends:	R (>= 2.9.2)
Built:	R 2.9.2; ; 2010-01-27 12:24:58 UTC; unix

Index:

`basicConfig`	`bootstrapping the logging package`
`addHandler`	`add a handler to a logger`
`getLogger`	`set defaults and get current values for a logger`
`removeHandler`	`remove a handler from a logger`
`setLevel`	`set logging.level for a logger`

to use this package, include logging instructions in your code, possibly like this:

```
library(logging)
basicConfig()
addHandler(writeToFile, logger="company", file="sample.log")
loginfo("hello world", logger="")
logwarn("hello company", logger="company.module")
```

the `basicConfig` function adds a console handler to the root logger, while the explicitly called `addHandler` adds a file handler to the 'company' logger. the effect of the above example is two lines on the console and just one line in the `sample.log` file.

the web pages for this package on r-forge are kept decently up to date and contain a usable tutorial. check the references.

Author(s)

Mario Frasca <mariotomo@gmail.com>

Maintainer: Mario Frasca <mario.frasca@nelen-schuurmans.nl>

References

the python logging module: http://docs.python.org/library/logging.html

this package at r-forge: http://logging.r-forge.r-project.org/

`addHandler` *add a handler to or remove one from a logger*

Description

use these functions to maintain the list of handlers attached to a logger.

addHandler and removeHandler are also offered as methods of the *Logger* S4 class. in that case there is no *logger* argument.

Usage

```
addHandler(handler, ..., logger = "")
removeHandler(handler, logger='')
```

Arguments

`handler`	the name of the handler, or its action
`...`	extra parameters for the action, to be stored in the handler list
`logger`	character parameter for the functional form: the name of the logger to which to attach the new handler, defaults to the root logger

Details

...may contain extra parameters that will be passed to the handler action. some elements in the ...will be interpreted here.

handler are implemented as environments. within a logger a handler is identified by its *name* and all handlers define at least the three variables:

level all records at level lower than this are skipped.

formatter a function getting a record and returning a string

action(msg, handler) a function accepting two parameters: a formatted log record and the handler itself. making the handler a parameter of the action allows us to have reusable action functions.

being an environment, a handler may define as many variables as you think you need. keep in mind the handler is passed to the action function, which can check for existence and can use all variables that the handler defines.

Examples

```
logReset()
addHandler(writeToConsole)
names(getLogger()[['handlers']])
loginfo('test')
removeHandler('writeToConsole')
logwarn('test')
```

`basicConfig` *bootstrapping the logging package*

Description

'basicConfig' and 'logReset' provide a way to put the logging package in a know initial state.

Usage

```
basicConfig(level)
logReset()
```

Arguments

`level`	the logging level of the root logger. defaults to INFO. please do notice that this has no effect on the handling level of the handler that basicConfig attaches to the root logger.

Details

basicConfig creates the root logger, attaches a console handler to it and sets the level of the handler to logging.level INFO. the level of the logger can be passed as a parameter to the function.

logReset reinitializes the whole logging system as if the package had just been loaded. typically, you would want to call basicConfig immediately after a call to logReset.

Examples

```
library(logging)
basicConfig()
logdebug("not shown, basic is INFO")
logwarn("shown and timestamped")
logReset()
logwarn("not shown, as no handlers are present after a reset")
```

getHandler *retrieves a handler from a logger.*

Description

handlers are not uniquely identified by their name. only within the logger to which they are attached is their name unique. this function is here to allow you grab a handler from a logger so you can examine and alter it.

typical use of this function is in setLevel(newLevel, getHandler(...)).

Usage

```
getHandler(handler, logger='')
```

Arguments

handler	the name of the handler, or its action.
logger	optional: the name of the logger. defaults to the root logger.

Examples

```
logReset()
addHandler(writeToConsole)
getHandler('writeToConsole')
```

getLogger *set defaults and get the named logger*

Description

make sure a logger with a specific name exists and return it as a *Logger* S4 object. if not yet present, the logger will be created and given the values specified in the ... arguments.

Usage

```
getLogger(name, ...)
```

Arguments

name	the name of the logger
...	any properties you may want to set in the newly created logger. these have no effect if the logger is already present.

Examples

```
getLogger()  # returns the root logger
getLogger('test.sub')  # constructs a new logger and returns it
getLogger('test.sub')  # returns it again
```

handler-actions *predefined handler actions*

Description

when you define a handler, you specify its name and the associated action. a few predefined actions are provided and this page documents them.

Usage

```
writeToFile(msg, handler, ...)
writeToConsole(msg, handler, ...)
```

Arguments

msg	the final formatted message to be output
handler	the handler owning this action
...	placeholder: accept any extra future parameter.

Details

a handler action is a function taking two parameters: the formatted log record and the handler to which the action is associated. the second parameter is useful so you can register the same handler action to handlers with different properties.

Examples

```
## define your own function and register it with a handler.
## author is planning a sentry client function.  please send
## any interesting function you may have written!
```

logging-internal *undocumented internal functions and variables*

Description

you hit this page because the function or variable has no real documentation page.

Examples

```
## no examples for undocumented functions
```

loginfo *entry points for logging actions*

Description

generate a log record and pass it to the logging system.

a log record gets timestamped and will be independently formatted by each of the handlers handling it.

leading and trailing whitespace is stripped from the final message.

Usage

```
logdebug(msg, ..., logger='')
loginfo(msg, ..., logger='')
logwarn(msg, ..., logger='')
logerror(msg, ..., logger='')
```

Arguments

msg	the textual message to be output, or the format for the ... arguments
...	if present, msg is interpreted as a format and the ... values are passed to it to form the actual message.
logger	the name of the logger to which we pass the record

Examples

```
logReset()
addHandler(writeToConsole)
loginfo('this goes to console')
logdebug('this stays silent')
```

loglevels *the logging levels, names and values*

Description

this list associates names to values and vice versa.

names and values are the same as in the python standard logging module.

Examples

```
loglevels['INFO']
loglevels['DEBUG']
```

setLevel *set logging.level for the object*

Description

alter an existing logger or handler, setting its logging.level to a new value. you can access loggers by name, while you must use getHandler to get hold of a handler.

Usage

```
setLevel(level, container)
```

Arguments

level	the new level for this object. can be numeric or character.
container	a logger or a handler.

Examples

```
basicConfig()
setLevel('FINEST')
setLevel('DEBUG', getHandler('basic.stdout'))
```

Index